UNIVERSITÀ DEGLI STUDI **ROMA TRE**

FACOLTÀ DI SS.MM.FF.NN.

SINTESI DELLA TESI DI LAUREA IN MATEMATICA

di

Rosina Otranto

Cifrari a Flusso e loro Crittoanalisi

RELATORE

Prof. Marco Pedicini

CANDIDATO

Rosina Otranto

Matr. 109871/29

ANNO ACCADEMICO 2004/2005

Maggio 2006

Classificazioni AMS: 68P25, 94A55, 11T71, 94A60.
Parole chiave: stream cipher, LFSR, crittonalisi, attacchi algebrici.

La crittografia studia i metodi che possono essere usati per inviare informazioni in forma celata, in modo tale che solamente il destinatario autorizzato possa rimuovere l'impedimento e leggere il messaggio in forma chiara.

Definizione 0.1 *Un crittosistema è una sestupla*

$$\{\mathcal{A}, \mathcal{M}, \mathcal{C}, \mathcal{K}, \{E_k : k \in \mathcal{K}\}, \{D_d : d \in \mathcal{K}\}\}.$$

formata da un alfabeto $\mathcal{A}$ di definizione, dall'insieme dei messaggi in chiaro $\mathcal{M}$, dall'insieme dei messaggi cifrati $\mathcal{C}$, dall'insieme delle chiavi $\mathcal{K}$ e da due famiglie di funzioni indicizzate sugli elementi di $\mathcal{K}$.

Per ogni elemento k di $\mathcal{K}$ è univocamente determinata una biezione da $\mathcal{M}$ a $\mathcal{C}$, denotata con E_k. E_k è denominata *funzione di cifratura* oppure *trasformazione di cifratura*. Si noti che la E_k deve essere una biiezione se si vuole che il processo sia invertito e sia recuperato un unico messaggio di testo in chiaro per ogni distinto messaggio di testo cifrato.
Per ogni $d \in \mathcal{K}$, D_d denota una biiezione da $\mathcal{C}$ a $\mathcal{M}$ (cioè $D_d : \mathcal{C} \to \mathcal{M}$). D_d è denominata *funzione di decifratura* o *trasformazione di decifratura*.

Se le chiavi k e d sono uguali, parliamo di cifrari a chiave simmetrica, altrimenti a chiave asimmetrica.

Con il termine *crittoanalista* si indica chi studia i punti deboli dei crittosistemi e quindi la *crittoanalisi* è la disciplina che studia i metodi di attacco ai crittosistemi.

Uno dei principi base della crittografia è il detto principio di Kerckhoffs, il quale asserisce che il grado di sicurezza che un certo sistema offre deve essere ottenuto dalla segretezza della chiave e non dalle caratteristiche del sistema, che devono essere supposte note a tutti compreso un eventuale attaccante.

Assumiamo quindi che chi voglia violare un sistema ne conosca le caratteristiche e che solo le chiavi (eventualmente non tutte) e il testo in chiaro (eventualmente non sempre) siano segreti.

I vari tipi di attacchi possono essere classificati come segue:

- *Ciphertext-only attack*: Un crittoanalista affronta la deduzione della chiave di decifratura o del testo in chiaro analizzando il solo testo cifrato. Ogni schema di cifratura che viene violato con tale attacco è insicuro. In realtà non può più essere definito sistema di cifratura;

- *Known-plaintext attack*: L'attaccante conosce uno o più testi in chiaro e le rispettive cifrature, da questi cerca di risalire alle chiavi o di decodificare altri testi cifrati;

- *Chosen-plaintext attack*: L'attaccante, scegliendo il testo in chiaro, è in grado di cifrare senza conoscere le chiavi. Le considerazioni che emergono da queste comparizioni sono utilizzate per decifrare il restante materiale cifrato;

- *Adaptative chosen-plaintext attack*: L'attaccante è in grado di cifrare ed è in grado di variare il testo in chiaro in conseguenza dei testi cifrati ottenuti. Non conosce le chiavi;

- *Chosen-ciphertext attack*: L'attaccante è in grado di decifrare un testo cifrato, ma non conosce le chiavi. Cerca di risalire alle chiavi.

- *Adaptive-chosen-attack*: è una variante dell'attacco con testo cifrato scelto dove la scelta del nuovo testo cifrato può dipendere dal testo in chiaro desunto dalla precedente analisi.

Chiaramente esistono anche altre possibilità, ad esempio, per *ricerca esaustiva della chiave* (o anche forza bruta), si intende il tentativo di decifrare il testo cifrato provando tutte le possibili chiavi di decifratura.

Il testo in chiaro corretto sarà uno dei pochi aventi senso compiuto che l'attaccante otterrà.

Nella crittoanalisi pratica più spesso un attacco è un metodo che serve a ridure lo spazio di ricerca nella crittoanalisi a forza bruta, infatti se solo si riuscisse a determinare un bit di informazione sulla chiave, la sicurezza nominale del crittosistema pari alla dimensione $|\mathcal{K}|$ del key-space risulterebbe dimezzata (e così anche il tempo necessario per l'attacco a forza bruta).

Ci sono due tipi di crittosistemi simmetrici: i cifrari a blocco (che cifrano un blocco di testo in chiaro in un blocco di testo cifrato, combinandolo in modo invertibile con una chiave fissa), e cifrari a flusso (che utilizzano una macchina a stati finiti, inizializzata con la chiave che produce una stringa di bit pseudocasuali, i quali sono combinati tramite un'operazione di or-esclusivo (XOR) con il testo in chiaro, da cui si ottiene il testo cifrato).

Un crittosistema che grantisce la sicurezza perfetta è il crittosistema OTP introdotto da C. Shannon, ma tale sistema è per sua natura di difficile utilizzo perché la sua sicurezza è basata sulla chiave, realmente aleatoria, di lunghezza pari al messaggio da cifrare; per questo motivo, il cifrario OTP non è utilizzato veramente nelle applicazioni ordinarie, ed invece sono stati concepiti i cifrari a flusso che sono costruiti come generatori di un flusso di chiave che si avvicini il più possibile ad un flusso aleatorio di informazione da utilizzare in uno schema tipo OTP.

In questa tesi dunque abbiamo trattato alcuni sistemi a flusso basati su *keystream generator*, cioè dei generatori di numeri pseudo-casuali (PRNG) realizzati mediante Linear Feedback Shift Register (LFSR), ovvero registri a scorrimento con retroazione lineare, studiandoli dal punto di vista della sicurezza rispetto a tre famiglie di attacchi: gli attacchi algebrici, gli attacchi di correlazione e gli attacchi di time/memory tradeoff.

Il quadro generale dei cifrari a flusso, che fanno uso degli shift register, può essere classificato come:

- Sincroni: in cui la sequenza cifrata è ottenuta dalla combinazione del keystream e del testo in chiaro,
  - Non lineari puri: la generazione della sequenza di output è basata su funzioni o procedure non lineari,
  - Combinazione di LFSR
    - LFSR puri: la sequenza è costruita utilizando un LFSR,
    - LFSR combinati

    - con memoria: la sequenza è costruita a partire dall'output di uno o più LFSR ma la combinazione degli output avviene utilizzando elementi della sequenza precedentemente calcolati e memorizzati in appositi registri per essere utilizzati successivamente,
    - semplici: gli output di uno o più LFSR vengono combinati tramite una opportuna funzione.

- Auto-sincroni: in cui la sequenza cifrata è ottenuta dalla combinazione del keystream, del testo in chiaro, e degli elementi di crittogramma precedetemente cifrati.

Noi ci occuperemo del primo caso, cifrari sincroni, dove il processo di cifratura è il seguente:

$$\sigma_{t+1} = f(\sigma_t, k)$$

$$z_t = g(\sigma_t, k)$$

$$c_t = h(z_t, m_t)$$

dove:

- σ_t, per $t = 0$, è lo stato iniziale e può dipendere dalla chiave k;
- f è la funzione dello stato successivo;
- g è la funzione che produce il keystream z_t, $t \geq 0$;
- h è la funzione di output che combina il keystream con il testo in chiaro, producendo il testo cifrato $c_t, t \geq 0$.

Ad ogni istante $t \geq 0$, il cifrario produce un nuovo carattere di keystream $z_t \in \mathbb{K}$, dove $\mathbb{K}$ è tipicamente un campo binario $\mathbb{F}_2$ o qualche estensione $\mathbb{F}_{2^w}$ del campo binario.

La lunghezza del carattere per l'algoritmo di cifratura a flusso è così definita avere w bit.

Un messaggio è diviso in N simboli e ogni simbolo è di lunghezza w bit:

$$m = m_0, m_1, \ldots, m_{N-1}, \quad m_t \in \mathcal{M}; \tag{1}$$

il messaggio viene cifrato simbolo per simbolo dalla funzione di output h.

L'output è una sequenza di simboli di testo cifrato $c = c_0, c_1, c_2, \ldots, c_{N-1}$ dove $c_t \in \mathcal{C}$. I cifrari sincroni utilizzano per generare numeri pseudocasuali delle macchine a stati finiti denominati *registri a scorrimento* o *shift registers*, noi ci occuperemo soprattutto degli LFSR, registri a scorrimento lineari.

Definizione 0.2 *Un registro a scorrimento di lunghezza L, è una macchina formata da L stati finiti, numerati da $0, 1, \ldots, L-1$; collegati in serie, e da un clock. Ogni stato è in grado di memorizzare la quantità minima di informazione: il bit, di avere un input ed un output; il clock controlla il movimento dei dati o contenuti degli stati. Denotiamo con R_i il singolo stato e con s_i il contenuto.*

Un registro a scorrimento di per sè non riesce a generare una sequenza di periodo massimo perché ad ogni impulso di clock il contenuto dello stato R_0 esce ed è un bit della sequenza degli output, il contenuto dello stato R_i passa nello stato R_{i-1} per ogni $1 \leq i \leq L-1$, lasciando libero quindi lo stato R_{L-1} del registro, da considerare in uno stato indefinito (né uno né zero) e di seguito tutti gli altri.

A tal proposito introduciamo una funzione di retroazione, ovvero una particolare funzione di alcuni bit del registro, che combinati tra loro in modo lineare, cioè utilizzando solo la somma binaria (somma modulo 2), ci permette di riempire lo stato del registro rimasto vuoto dopo lo spostamento, questo nuovo bit immesso prende il nome di bit di retroazione o *feedback*.

Lo stato iniziale dell'LFSR è $[s_{L-1}, \ldots, s_1, s_0]$, allora la sequenza $s = s_0, s_1, s_2 \ldots$ è determinata dalla seguente formula di ricorrenza:

$$s_j = a_1 s_{j-1} + a_2 s_{j-2} + \ldots + a_L s_{j-L} \; mod(2) \; per \; j \geq L,$$

dove $a_1, a_2, \ldots, a_L$ sono coefficienti in $\mathbb{K}$ del polinomio di feedback:

$$f(x) = 1 + a_1 x + a_2 x^2 + \ldots + a_L x^L,$$

I coefficienti $a_1, a_2, \ldots a_L$ valgono 0 oppure 1, precisamente $a_i = 0$ se lo stato R_i non partecipa a riempire l'ultimo stato; mentre $a_i = 1$ nel caso contrario.

L'insieme dei bit che fanno parte del polinomio di feedback viene chiamato *tap sequence.*

Chiamiamo *stato del registro* al tempo t l'insieme dei contenuti degli stati al tempo t.

I registri a scorrimento per loro natura hanno una relazione ben precisa tra stati interni e periodo della sequenza generata infatti, se $f(x)$ è un polinomio primitivo e irriducibile, allora il registro emette una sequenza di output con periodo massimo $2^L - 1$, dove L è il grado del polinomio.

Proposizione 0.1 *Se consideriamo un registro R lineare di lunghezza L e periodo massimo $2^L - 1$, anche se si cambia la configurazione iniziale, il periodo resta massimo.*

Da quanto detto segue che il periodo massimo di una sequenza pseudocasuale cresce esponenzialmente con la lunghezza del registro, di conseguenza le sequenze, per L grande, possono essere considerate aperiodiche ai fini pratici.

Ogni registro a scorrimento lineare, prima di essere usato come generatore di sequenze pseudocasuali deve essere inizializzato, ovvero bisogna introdurre la sequenza di uno e di zero da cui il generatore deve partire per generare la sequenza pseudocasuale. Tale valore, insieme al polinomio di retroazione $f(x)$, costituisce la chiave segreta che serve al mittente e al destinatario per ricostruire la sequenza pseudocasuale necessaria per le operazioni di cifratura e decifratura del messaggio in chiaro; quindi al variare di uno o di entrambi questi elementi , varia la sequenza degli output.

Dunque, per cambiare la chiave, si può cambiare lo stato iniziale ed usare la stessa funzione, oppure cambiare la funzione, oppure l'uno e l'altro.

Definizione 0.3 *Una sequenza è pseudocasuale se soddisfa le seguenti proprietà (dette* ***postulati di Golomb****):*

- *Il numero di 0 in una sequenza è approssimativamente uguale al numero di 1;* $L/2$ *se* L *è pari,* $(L+1)/2$ *se* L *è dispari.*
- *Metà dei run (sottosequenza della sequenza costituita da elementi consecutivi uguali) ha lunghezza pari ad* 1*, un quarto ha lunghezza pari a* 2 *e così via. Un esempio potrebbe essere la sequenza:* 10101010100110011.

Gli LFSR non forniscono sequenze pseudocasuali utili dal punto di vista crittografico. Infatti se un attaccante sa che il testo cifrato è stato ottenuto usando un registro a scorrimento di lunghezza L e riesce a scoprire $2L$ bit di testo in chiaro ed i corrispondenti $2L$ bit di testo cifrato, può ricostruire la chiave facilmente, perché deve solo risolvere in $\mathbb{K}$ un sistema lineare di L equazioni in L incognite.

Per distruggere la proprietà di linearità dei registri a scorrimento è utile combinare questi registri con una funzione di retroazione non lineare, ed ottenere due casi:

1. LFSR combinati con memoria;
2. LFSR combinati senza memoria, o semplici.

Uno dei modi per ottenere combinazioni non lineari delle uscite di diversi LFSR è considerare una serie di LFSR in parallelo non necessariamente della stessa lunghezza (non linear combination generator). Le uscite di questi registri sono convogliate in una funzione non lineare f (combining function) il cui scopo è quello di creare un'uscita crittograficamente sicura.

La funzione di combinazione deve essere scelta in maniera tale che non si evidenzi una dipendenza statistica tra una porzione di sequenza di un qualsiasi LFSR, appartenente al generatore, ed il keystream di uscita.

Lo scopo della *crittoanalisi* è proprio quello di riuscire ad attaccare cifrari ritenuti sicuri.

La crittoanalisi degli LFSR è basata su due operazioni principali:

1. l'algoritmo di Berlekamp-Massey, utile per ricavare il polinomio di feedback a partire dal keystream; questo si può, a seconda dei casi, effettuare utilizzando l'algoritmo lineare (tramite una semplice inversione di matrice) oppure con l'algoritmo di Massey.

2. Calcolo delle radici di un polinomio su $\mathbb{F}_q$, utilizzando l'algoritmo di Berlekamp.

1. • *L'algoritmo di Berlekamp-Massey*:
   Questo algoritmo produce il polinomio minimo dopo un gran numero di passi finiti. In particolare, se si conosce un testo in chiaro e se ne osserva la versione cifrata, una parte di keystream è ottenuto con l'operazione $m_i \oplus c_i$ (dove m_i sono i simboli di testo in chiaro e c_i quelli cifrati). Se tale keystream è usato per cifrare altri messaggi, o se il materiale a disposizione è un frammento del testo da decifrare, è possibile ricostruire il polinomio irriducibile che determina la funzionalità dell'LFSR utilizzato e che, inizializzato con qualsiasi stringa di bit di lunghezza L, riproduce l'intera sequenza.
   Sia $s_0, s_1, \ldots$ una sequenza di elementi in $\mathbb{F}_q$ che genera la funzione:
   $$G(x) = \sum_{n=0}^{\infty} s_n x^n.$$
   Per $j = 0, 1, \ldots$ definiamo il polinomio $g_j(x)$ e $h_j(x)$ su $\mathbb{F}_q$, interi m_j e b_j di $\mathbb{F}_q$ come segue. Inizialmente poniamo:
   $$g_0(x) = 1,\ h_0(x) = x,\ m_0 = 0.$$
   Quindi continuiamo in modo ricorsivo, lasciando che b_j sia il coefficiente di x^j in $g_j(x)G(x)$ e poniamo:
   $$g_{j+1}(x) = g_j(x) - b_j h_j(x)$$
   $$h_{j+1}(x) = \begin{cases} b_j^{-1} x g_j(x) & \textit{se } b_j \neq 0 \textit{ e } m_j \geq 0 \\ x h_j(x) & \textit{altrimenti} \end{cases}$$

$$m_{j+1} = \begin{cases} -m_j \;\; se\ b_j \neq 0\ e\ m_j \geq 0 \\ m_j + 1 \quad altrimenti \end{cases}$$

Se $s_0, s_1 \ldots$ è una sequenza lineare omogenea ricorsiva con un polinomio minimo di grado k, ne segue che $g_{2k}(x)$ è dato da :

$$m(x) = x^k g_{2k}\left(\frac{1}{x}\right).$$

Se si sa solo che il polinomio minimo è di grado $\leq k$, allora si pone $r = \lfloor k + \frac{1}{2} - \frac{1}{2}m_{2k} \rfloor$, dove $\lfloor y \rfloor$ denota il più grande intero $\leq y$, e il polinomio minimo $m(x)$ è dato da:

$$m(x) = x^r g_{2k}\left(\frac{1}{x}\right).$$

In entrambi i casi, come si può vedere immediatamente dall'algoritmo, $m(x)$ dipende solo da $2k$ termini $s_0, s_1, \ldots, s_{2k-1}$ della sequenza. Quindi si può sostituire la funzione generatrice $G(x)$ nell'algoritmo con il polinomio

$$G_{2k}(x) = \sum_{n=0}^{2k-1} s_n x^n.$$

- Oppure possiamo calcolare *i coefficienti del polinomio di feedback con la matrice inversa*:
  In generale il keystream è prodotto da m-ple iniziali: $(z_i, \ldots, z_m)$, usando la ricorrenza lineare:

$$z_{m+i} = \sum_{j=0}^{m-1} c_j z_{i+j}\ mod\ 2 \;\; i \geq 1$$

che è un'equazione lineare in m incognite, dove $c_0, \ldots, c_{m-1} \in \mathbb{Z}_p$. Da questa possiamo ricavare l'intera sequenza di keystream.

Supponiamo, dunque, di conoscere una stringa di testo in chiaro $x_1 x_2 \ldots x_n$ e la corrispondente stringa di testo cifrato $y_1 y_2 \ldots y_n$. Allora possiamo calcolare i bit di keystream $z_i = (x_i + y_i)\ mod\ 2$ con $1 \leq i \leq n$. Inoltre supponiamo di conoscere il valore di m. Ci

rimane solo da calcolare $c_0, \ldots, c_{m-1}$ per poter ricostruire l'intero keystream. In altre parole, si devono poter determinare i valori delle m incognite.

Se $n \geq 2m$, allora esistono m equazioni lineari in n incognite, che possono essere successivamente risolte.
Il sistema di m equazioni lineari può essere scritto in forma matriciale dalla seguente forma:

$$(z_{m+1}, z_{m+2}, \ldots, z_{2m}) = (c_0, c_1, \ldots, c_{m-1}) \begin{pmatrix} z_1 & z_2 & \cdots & z_m \\ z_2 & z_3 & \cdots & z_{m+1} \\ \vdots & \vdots & \ldots & \vdots \\ z_m & z_{m+1} & \cdots & z_{2m-1} \end{pmatrix}.$$

Se la matrice dei cefficienti ha un'inversa (cioè determinante > 0), otteniamo la soluzione:

$$(c_0, c_1, \ldots, c_{m-1}) = (z_{m+1}, z_{m+2}, \ldots, z_{2m}) \begin{pmatrix} z_1 & z_2 & \cdots & z_m \\ z_2 & z_3 & \cdots & z_{m+1} \\ \vdots & \vdots & \ldots & \vdots \\ z_m & z_{m+1} & \cdots & z_{2m-1} \end{pmatrix}^{-1}.$$

In realtà la matrice ha un'inversa se m è il grado della relazione di ricorrenza usata per generare il keystream.

2. *Algoritmo di Berlekamp*:
   Questo algoritmo permette di fattorizzare polinomi di grado abbastanza elevato su campi finiti.

   Nel nostro caso come campi finiti consideriamo $\mathbb{F}_p[x]$, cioè polinomi con coefficienti nel campo di p elementi, p primo. Una fattorizzazione di $f(x)$ si potrebbe ottenere anche attraverso l'algoritmo delle successioni successive, cioè possiamo controllare tutti i polinomi di grado $\leq \frac{d}{2}$ ($d = deg(f(x))$) come possibili fattori di $f(x)$. Ovviamente, non appena d è abbastanza elevato, questo procedimento risulta molto lento.

Teorema 0.1 (Teorema di Berlekamp) *Sia* $f(x) \in \mathbb{F}_p[x]$ *di grado* d. *Sia* $\mathcal{Q}$ *la matrice* $d \times d$ *dove la* i*-esima riga è il vettore dei coefficienti del resto polinomiale* $r_i(x)$, *e* $x^{ip} = f(x)q_i(x) + r_i(x)\ \forall\ i = 0, \ldots, d-1$. *Sia* $b = (b_0, \ldots, b_{d-1})$ *una soluzione di* $b(\mathcal{Q} - \mathcal{I}) = 0$ *e sia* $g(x) = b_0 + b_1 x + \ldots + b_{d-1}x^{d-1}$.
Se $g(x)$ *ha grado* $\geq 1 \Rightarrow \forall s \in \mathbb{F}_p[x]$ $g(x) - s$ *e* $f(x)$ *hanno un fattore comune di grado* ≥ 1.

Dopo aver illustrato le operazioni principali della crittoanalisi, trattiamo, in maniera più dettagliata, alcuni attacchi che utilizzano queste operazioni.

I L'attacco algebrico è un metodo piuttosto nuovo per il crittoanalista anche se le idee di base sono state conosciute in letteratura matematica da tempo.

Questo attacco sfrutta la struttura algebrica intrinseca del cifrario trattato, ed esprime la trasformazione di cifratura come un insieme di equazioni polinomiali multivariate, successivamente, si prova a risolvere tale sistema per ottenere la chiave di cifratura.

Il problema di un crittoanalista, può essere descritto come segue.

Sia $(k_0, \ldots, k_{n-1})$ lo stato iniziale, allora l'output del cifrario (cioè il keystream) sarà:

$$\begin{cases} z_0 = f(k_0, \ldots, k_{n-1}) \\ z_1 = f(L(k_0, \ldots, k_{n-1})) \\ z_2 = f(L^2(k_0, \ldots, k_{n-1})) \\ \vdots \end{cases}$$

Il problema è recuperare la chiave $k = (k_0, \ldots, k_{n-1})$ da un certo sottoinsieme di bit z_i di keystream.

Per poter applicare l'attacco algebrico dobbiamo considerare solo i cifrari a flusso sincroni, con un clock regolare, oppure con un clock il cui movimento sia conosciuto. Per semplicità restringiamo lo studio

a cifrari binari, in cui lo stato ed il keystream sono composti di sequenze di bit, e questi generano un bit alla volta. Ci limitiamo al caso in cui la funzione che calcola lo stato successivo, sia lineare su $GF(2)$.

Chiamiamo L questa funzione e supponiamo che L sia pubblica e che solo lo stato sia segreto. Inoltre si suppone che la funzione f, che calcola i bit di output dallo stato, sia pubblica e che non dipenda dalla chiave segreta del cifrario. Questa funzione f è denominata *funzione filtrante non lineare*.

Realizziamo l'attacco conoscendo parzialmente il testo in chiaro, ossia conoscendo alcuni bit del testo in chiaro e i corrispondenti bit di testo cifrato. I bit non devono essere consecutivi.

Cerchiamo una f che abbia buona correlazione con funzioni multivariate non lineari di grado basso. Oppure che:

- $S1$: La funzione Boooleana f ha un basso grado algebrico D (criterio classico);
- $S2$: oppure la f può essere approssimata da una tale funzione con una probabilità di circa 1 (nuovo criterio), questa probabilità è solitamente denotata con $1 - \varepsilon$ con ε piccola;
- $S3$: il polinomio multivariato f ha qualche multiplo fg di basso grado d, dove g è un certo polinomio multivariato non nullo;
- $S4$: la f ha qualche multiplo fg, tale che fg può approssimarsi ad una funzione di basso grado con una certa probabilità $(1 - \varepsilon)$.

Il metodo di base per realizzare l'attacco è stato introdotto, in un documento, da Kipnis e da Shamir [27], dove introducono una tecnica denominata *linearizzazione*.

Con questo metodo si considerano tutti i monomi del sistema come variabili indipendenti e si prova a risolvere il sistema con queste nuove variabili con le tecniche di algebra lineare. Per poter applicare il meto-

do, è necessario che il numero di equazioni linearmente indipendenti nel sistema sia, approssimativamente, uguale alle incognite nel sistema.

Quando questa condizione non si verifica, esistono alcune tecniche che permottono di generare delle equazioni linearmente indipendenti, per poter, così, applicare la tecnica di linearizzazione.

Il metodo classico che viene usato è la procedura di Buchberger usando le basi di Gröbner.

Tale metodo, utilizzato per risolvere il sistema di equazioni quadratiche, è valido in molti casi, ma i test di verifica per successo e complessità non sono ben chiari.

In seguito, Courtois ed altri hanno dato un'analisi teorica e pratica della procedura di Buchberger ed hanno esteso il lavoro con un'altro algoritmo conosciuto come *XL-algoritmo esteso*, il quale è basato sulle "basi" di Gröbner e sulla "linearizzazione di grado".

Se A è un sistema di m equazioni quadratiche f_i in n variabili sul campo $\mathbb{K}$ e $D \in \mathbb{N}$, l'algoritmo XL esegue le seguenti operazioni:

- *Prodotto*: genera tutti i prodotti $\prod_{j=1}^{k} x_{ij} \times f_i \quad \text{con } k \leq D-2$; dove D è il grado del sistema e 2 è il grado dell'equazione quadratica f_i.
- *Linearizzazione*: considera ogni monomio di grado $\leq D$ come una nuova variabile ed esegue l'eliminazione gaussiana sul sistema ottenuto al passo precedente;
- *Risoluzione*: se si genera almeno un'equazione con una variabile, al passo precedente, allora l'algoritmo risolve questa equazione;
- *Ripetizione*: semplifica le equazioni e ripete per trovare i valori delle altre variabili.

Oltre all'algoritmo XL vedremo anche l'algoritmo *XSL*, che è basato sul metodo di XL, ma sfrutta la struttura specifica delle equazioni e il fatto

che queste siano *sparse* [1]; invece di moltiplicare le equazioni per tutti i monomi di grado $\leq D-2$, in XSL le equazioni sono moltiplicate solo per dei monomi particolari, cioè dei monomi che già compaiono nell'equazione.

Vedremo due algoritmi A e B per un attacco algebrico veloce:

Algoritmo **A** :

(a) si sceglie un'appropriata chiave $\widehat{\mathcal{K}}$ e si calcola

$$\widehat{z_t} = F_t(\widehat{\mathcal{K}}) \; per \; t = 1, \ldots, 2T.$$

(b) Si applica l'algoritmo di Berlekamp-Massey, per cercare $c_0, \ldots, c_{T-1}$, con

$$\sum_{i=0}^{T-1} c_i \cdot F_{t+1}(\widehat{\mathcal{K}}) = 0, \; \forall t. \tag{2}$$

Algoritmo **B** :
Dati:

- polinomi primitivi a due a due coprimi $min_1(x), \ldots, min_k(x)$,
- una funzione Boooleana arbitraria F,
- una partizione $F = F_1 + \ldots + F_l$ tale che i polinomi minimi, $min(F_i)$ sono a due a due coprimi.

Operazioni:

- trovare $min(F)$.

Algoritmo:

[1] Sia t il numero di monomi che compaiono nel sistema, sia d il grado di una determinata equazione del sistema, sia m il numero di equazioni che formano il sistema e s il numero di variabili, allora diciamo che le equazioni sono *sparse* se si ha $t << \begin{pmatrix} s \\ d \end{pmatrix}$ e si dice che il sistema è *sovradefinito* se $r >> s$

- calcolare i polinomi minimi $min(F_i)$, usando l'algoritmo A.

  Ciò può essere fatto in parallelo. Se $F_i = G_1 \cdot G_2$ con $min(G_1)$ e $min(G_2)$ coprimi, allora possiamo usare l'algoritmo $\otimes$-prodotto.
- Calcolare $min(F) = min(F_1) \cdot \ldots \cdot min(F_l)$.

II Un'altro tipo di attacco interessante è *l'attacco di correlazione*. Questo tipo di attacco è applicato a quei generatori di keystream che combinano, mediante un'oppurtuna funzione, le uscite dei diversi generatori, ad esempio di tipo LFSR.

Tale attacco è mirato ad isolare qualche debolezza "nel modo" in cui la funzione di combinazione unisce i vari contributi e crea un flusso di uscita in cui è possibile il contributo di qualche ingresso, definito debole. In tali casi esiste una correlazione tra la sequenza di uscita e una sequenza interna che può essere analizzata in dettaglio prima di occuparsi delle altre.

Siano dati $R_1, R_2, \ldots, R_n$ LFSR di massima lunghezza (ove cioè la connessione polinomiale è una primitiva di grado L) e ciascuna lunghezza sia indicata con $L_1, L_2, \ldots, L_n$.

Tali LFSR sono elementi costitutivi di un generatore a combinazione non lineare. Se sono note la funzione di combinazione e la connessione polinomiale di ciascun registro, il numero di differenti chiavi, ovvero di registri è:

$$\prod_{i=1}^{n}(2^{L_i} - 1).$$

Supponiamo che esista una correlazione tra il keystream e la sequenza in uscita da R_1 con probabilità di correlazione $p > \frac{1}{2}$.

Se si ha a disposizione un sufficiente frammento di keystrem (ottenibile mediante un attacco known-plaintext), lo stato iniziale di R_1 può essere dedotto valutando il numero di coincidenze tra il keystream e tutte le possibili repliche traslate della sequenza in uscita da R_1, fino a che si ottiene un numero di coincidenze con probabilità di correlazione p.

In queste condizioni trovare lo stato iniziale di R_1 richiede al massimo $2^{L_1} - 1$ tentativi.

Nei casi in cui esiste una correlazione tra il keystream e le sequenze in uscita di ciascun $R_1, R_2, \ldots, R_n$ LFSR ciascun seme di inizializzazione può essere determinato indipendentemente in un numero totale di tentativi pari a:

$$\sum_{i=1}^{n}(2^{L_i} - 1).$$

In generale per rendere la struttura del generatore resistente all'attacco di correlazione, ci si deve assicurare che non ci sia nessuna dipendenza statistica tra ogni piccolo sottoinsieme delle sequenze del sottogeneratore R_n e la sequenza di keystream.

Pertanto è utile fare uso di funzioni che abbiano immunità da correlazione:

Supponiamo che gli n sottogeneratori siano fonti binarie simmetriche indipendenti, cioè che la sequenza $x_{i1}, x_{i2}, x_{i3}, \ldots$ prodotta dal sottogeneratore R_i sia una sequenza di variabili casuali binarie indipendenti identicamente distribuite ugualmente probabile ad essere o zeri o uni e indipendenti dalle sequenze prodotte da altri sottogeneratori. La sequenza di keystream $z_1, z_2, z_3, \ldots$ è determinata da:

$$z_j = f(x_{1j}, x_{2j}, \ldots, x_{nj}).$$

Segue dalle ipotesi che la sequenza di keystream $z_1, z_2, z_3, \ldots$ è anche indipendente identicamente distribuita (i.i.d) ma non necessariamente bilanciata.

Supponiamo che $x_j = (x_{1j}, x_{2j}, \ldots, x_{nj})$ sia un'n-pla dell'output dei sottogeneratori al tempo j. Diremo che la funzione di combinazione f ha immunità da correlazione di ordine m-esimo se ogni m-pla, ottenuta scegliendo m componenti da x_j è statisticamente indipendente da z_j per ogni $j = 1, 2, 3, \ldots$.

Dall'invarianza del tempo del sistema e dalla natura i.i.d. di tutte le sequenze, ciò equivale a dire che ogni sottoinsieme di variabili casuali m scelte da $x_1, x_2, \ldots, x_n$, è statisticamente indipendente da:

$$z = f(x_1, x_2, \ldots, x_n), \tag{3}$$

quando $x_1, x_2, \ldots, x_n$ sono variabili casuali binarie i.i.d. bilanciate. Quindi f ha immunità da correlazione di m-esimo ordine se e solo se per ogni scelta di indici $i_1, i_2, \ldots, i_m$ con $1 \leq i_1 < i_2 < \ldots < i_m \leq n$, la variabile random z di (3) è statisticamente indipendente dal vettore random $(x_{i_1}, x_{i_2}, \ldots, x_{i_m})$.

Supponendo che funzione binaria f di n variabili casuali binarie può essere scritta nella sua forma algebrica normale, cioè come la somma di prodotti su GF(2):

$$f(x_1, x_2, \ldots, x_n) = \tag{4}$$

$$= a_0 + a_1 x_1 + a_2 x_2 + \ldots + a_n x_n + a_{12} x_1 x_2 + a_{13} x_1 x_3 + \ldots + a_{12\ldots n} x_1 x_2 \ldots x_n.$$

Teorema 0.2 *Se $f(x_1, x_2, \ldots, x_n)$ ha immunità da correlazione di m-esimo ordine, dove $1 \leq m < n$, allora nessun prodotto di $n - m + 1$ di più variabili può essere presente nella forma algebrica normale (4) di f. Inoltre se:*

$$P[f(x_1, x_2, \ldots, x_n) = 1] = P[f(x_1, x_2, \ldots, x_n) = 0]$$

quando $x_1, x_2, \ldots, x_n$ sono variabili random binarie i.i.d, allora nessun prodotto di $n - m$ variabili può essere presente nella forma algebrica normale di f a meno che $m = n - 1$.

Si noti che, per $m = n - 1$, tutti i prodotti di ordine $n - m = 1$ devono stare in f. Così le uniche funzioni possibili con immunità di correlazione di $(n-1)$-esimo ordine sono:

$$f(x) = x_1 + x_2 + \ldots + x_n + c$$

dove $c = 1$ *oppure*; $c = 0$.

III Un altro attacco che abbiamo visto è l'attacco *tradeoff*.

Per molti algoritmi di cifratura possiamo fare un tradeoff (compromesso) per la complessità del tempo, della memoria e dei dati, nel caso di una ricerca esaustiva della chiave.

Tale attacco ha due fasi:

- di *preprocessing* (che richiede molto tempo) in cui l'attaccante esplora la struttura generale del crittosistema e ricapitola i suoi risultati in grandi tabelle o *table lookup* (che non sono legate a particolari chiavi).

  In queste tabelle il crittoanalista prima cifra un fissato testo in chiaro P_0 con ciascuna delle N chiavi e produce N testi cifrati, poi questi testi cifrati vengono fascicolati e immagazinati con le corrispondenti chiavi collegate;
- di *realtime* (tempo reale), durante la quale l'attaccante conosce i dati reali redatti da una particolare chiave sconosciuta, e il suo obiettivo è di usare le tabelle precalcolate per trovare la chiave il più rapidamente possibile.

In tutto l'attacco tradeoff time/memory ci sono 5 parametri chiave:

- N: è la grandezza dello spazio di ricerca;
- P: è il tempo richiesto nella fase di preprocessing dell'attacco;
- M: è la quantità di memoria di accesso random (sotto forma di hard disk o DVD) disponibile dall'attaccante;
- T: è il tempo richiesto nella fase di realtime dell'attacco;
- D: è la quantità di dati in tempo reale disponibile dall'attaccante.

Il migliore attacco tradeoff Time/memory conosciuto è dovuto ad Hellman, il quale usa tutte le combinazioni di parametri che soddisfano i seguenti rapporti:

- $TM^2 = N^2$;
- $P = N$;
- $D = 1$;

Affrontiamo il problema di trovare l'inversa della funzione f, ossia la funzione applicata alla chiave per ottenere il keystream con un algoritmo, il quale è più veloce della ricerca esaustiva.

Secondo Hellman bisogna utilizzare una fase di preprocessing, dove si calcolano N punti e li si rappresentano su una matrice rettangolare $m \times t$, le cui righe rappresentano l'iterazione della funzione f t-volte su m punti iniziali scelti casualmente.

L'output della fase di preprocessing è l'insieme di coppie (punto iniziale, punto finale) di tutti i percorsi scelti, in ordine crescente. Durante l'attacco reale ci viene dato un valore z, e ci viene richiesto di trovare il predecessore k sotto la f. Se questa k è raggiunta da uno dei percorsi precalcolati, l'algoritmo applica ripetutamente la f ad z fino a che essa raggiunge il punto finale immagazzinato, salta al suo punto iniziale associato e applica ripetutamente la f ad un punto iniziale fino a che esso raggiunge ancora z. Il punto precedente che visita è la k desiderata.

Fra gli attacchi e i crittosistemi che abbiamo affrontato illustriamo, per esempio, l'attacco *algebrico* applicato al crittosistema *Toyocrypt*.

Il cifrario a flusso Toyocrypt, il quale ha un LFSR di
128-bit, e così $n = 128$. La funzione Booleana è della forma

$$f(s_0, \ldots, s_{127}) = s_{127} + \sum_{i=0}^{62} s_i s_{\alpha_i} + s_{10}s_{23}s_{32}s_{42} +$$

$$+ s_1 s_2 s_9 s_{12} s_{18} s_{20} s_{23} s_{25} s_{26} s_{28} s_{33} s_{38} s_{41} s_{42} s_{51} s_{53} s_{59} + \prod_{i=0}^{62} s_i$$

essendo $\{\alpha_0, \ldots, \alpha_{62}\}$ una certa permutazione dell'insieme $\{63, \ldots, 125\}$.

Questo sistema è abbastanza vulnerabile ad un attacco che usa le approssimazioni di ordine basso , c'è solo un monomio di grado 17 e uno di grado 63. I monomi di ordine più alto sono quasi sempre nulli.

Usiamo il seguente teorema per costruire la funzione Booleana:

Teorema 0.3 *Sia g una funzione Booleana $g : GF(2)^{2k} \rightarrow GF(2)$. Tutte le funzioni g possono essere ricondotte a:*

$$f(x_1, x_2, \ldots, x_{2k}) = x_1x_2 + x_3x_4 + \ldots + x_{2k-1}x_{2k} + g(x_1, x_2, x_{2k-1}).$$

La funzione di Toyocrypt è appunto uno xor di s_{127} e una funzione costruita in base al teorema.

Nel nostro attacco dobbiamo trovare la g, tale che fg sia di grado basso, seguendo i principi dei casi S_3 (o S_4). Come trovare questa funzione g tale che fg sia di basso grado? Consideriamo i termini di alto grado in $f(s)$ (trascurando i termini di grado più basso), vediamo se essi sono divisibili per un fattore comune di grado più basso $g'(s)$.

Allora (per polinomi sul campo $GF(2)$), osserviamo che $f(s) \times g(s)$ con $g(s) = g'(s) - 1$ è di grado basso. Nel caso di Toyocrypt, osserviamo che i termini di grado 4, 17 e 63 sono divisibili per un fattore comune $s_{23}s_{42}$. Per ogni bit z_t di keystream, partiamo dall'equazione $f(s) = z_t$, e moltiplichiamo entrambi i lati per $g(s) = (s_{23} - 1)$. Otteniamo $f(s)s_{23} - f(s) = z_t(s_{23} - 1)$. I monomi divisibili per s_{23} in f si cancelleranno, ciò che rimane è un'equazione di grado 3, vera con probabilità 1. Ripetiamo lo stesso procedimento per s_{42}, cioè metteremo $g(s) = s_{42} - 1$. Da ciò, otterremo un semplice attacco di linearizzazione che segue il caso S_3. Per ogni bit di keystream, otteniamo due equazioni di grado 3 in s_i e quindi due equazioni di grado 3 in k_i. La linearizzazione lavorerà non appena $R > T$ e otterremo $T \approx \begin{pmatrix} n \\ d \end{pmatrix} = \begin{pmatrix} 128 \\ 3 \end{pmatrix}$ monomi. Si ha $R = 2m$, e sarà sufficiente avere $m = \frac{T}{2} = 2^{17,4}$ bit di keystream.

Bibliografia

[1] Frederik Armknecht and Matthias Krause. Algebraic attacks on combiners with memory. In *Advances in cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Comput. Sci.*, pages 162–175. Springer, Berlin, 2003.

[2] Gwénolé Ars, Jean-Charles Faugère, Hideki Imai, Mitsuru Kawazoe, and Makoto Sugita. Comparison between XL and Gröbner basis algorithms. In *Advances in cryptology—ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Comput. Sci.*, pages 338–353. Springer, Berlin, 2004.

[3] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In *Progress in cryptology—INDOCRYPT 2000 (Calcutta)*, volume 1977 of *Lecture Notes in Comput. Sci.*, pages 43–51. Springer, Berlin, 2000.

[4] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In *Advances in cryptology—ASIACRYPT 2000 (Kyoto)*, volume 1976 of *Lecture Notes in Comput. Sci.*, pages 1–13. Springer, Berlin, 2000.

[5] P. Camion, C. Carlet, P. Charpin, and N. Sendrier. On correlation-immune functions. In *Advances in cryptology—CRYPTO '91 (Santa Barbara, CA, 1991)*, volume 576 of *Lecture Notes in Comput. Sci.*, pages 86–100. Springer, Berlin, 1992.

[6] Vladimir Chepyzhov and Ben Smeets. On a fast correlation attack on certain stream ciphers. In *Advances in cryptology—EUROCRYPT '91 (Brighton, 1991)*, volume 547 of *Lecture Notes in Comput. Sci.*, pages 176–185. Springer, Berlin, 1991.

[7] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in cryptology—EUROCRYPT 2000 (Bruges)*, volume 1807 of *Lecture Notes in Comput. Sci.*, pages 392–407. Springer, Berlin, 2000.

[8] Nicolas T. Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Advances in cryptology—CRYPTO 2003*, volume 2729 of *Lecture Notes in Comput. Sci.*, pages 176–194. Springer, Berlin, 2003.

[9] Nicolas T. Courtois. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt. In *Information security and cryptology—ICISC 2002*, volume 2587 of *Lecture Notes in Comput. Sci.*, pages 182–199. Springer, Berlin, 2003.

[10] Nicolas T. Courtois and Willi Meier. Algebraic attacks on stream ciphers with linear feedback. In *Advances in cryptology—EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Comput. Sci.*, pages 345–359. Springer, Berlin, 2003.

[11] Nicolas T. Courtois and Jacques Patarin. About the XL algorithm over GF(2). In *Topics in cryptology—CT-RSA 2003*, volume 2612 of *Lecture Notes in Comput. Sci.*, pages 141–157. Springer, Berlin, 2003.

[12] Ed Dawson, Leonie Simpson, and Jovan Golić. A survey of divide and conquer attacks on certain irregularly clocked stream ciphers. In *Cryptography and computational number theory (Singapore, 1999)*, volume 20 of *Progr. Comput. Sci. Appl. Logic*, pages 165–185. Birkhäuser, Basel, 2001.

[13] Patrik Ekdahl. *On LFSR based Stream Ciphers.* PhD thesis, Lund university, 2003.

[14] Jean-Charles Faugére. A new efficient algorithm for computing Gröbner bases (F_4). *J. Pure Appl. Algebra*, 139(1-3):61–88, 1999. Effective methods in algebraic geometry (Saint-Malo, 1998).

[15] Eric Filiol and Caroline Fontaine. Highly nonlinear balanced Boolean functions with a good correlation-immunity. In *Advances in cryptology—EUROCRYPT '98 (Espoo)*, volume 1403 of *Lecture Notes in Comput. Sci.*, pages 475–488. Springer, Berlin, 1998.

[16] Scott Fluhrer and Stefan Lucks. Analysis of the E_0 encryption system. In *Selected areas in cryptography*, volume 2259 of *Lecture Notes in Comput. Sci.*, pages 38–48. Springer, Berlin, 2001.

[17] J. Dj. Golić, M. Salmasizadeh, L. Simpson, and E. Dawson. Fast correlation attacks on nonlinear filter generators. *Inform. Process. Lett.*, 64(1):37–42, 1997.

[18] Jovan Dj. Golić. Linear statistical weakness of alleged RC4 keystream generator. In *Advances in cryptology—EUROCRYPT '97 (Konstanz)*, volume 1233 of *Lecture Notes in Comput. Sci.*, pages 226–238. Springer, Berlin, 1997.

[19] Jovan Dj. Golić, Vittorio Bagini, and Guglielmo Morgari. Linear cryptanalysis of Bluetooth stream cipher. In *Advances in cryptology—EUROCRYPT 2002 (Amsterdam)*, volume 2332 of *Lecture Notes in Comput. Sci.*, pages 238–255. Springer, Berlin, 2002.

[20] Jovan Dj. Golić and Renato Menicocci. Edit probability correlation attacks on stop/go clocked keystream generators. *J. Cryptology*, 16(1):41–68, 2003.

[21] Jovan Dj. Golić and Renato Menicocci. Correlation analysis of the alternating step generator. *Des. Codes Cryptogr.*, 31(1):51–74, 2004.

[22] Jovan Dj. Golić and Luke O'Connor. A cryptanalysis of clock-controlled shift registers with multiple steps. In *Cryptography: policy and algorithms (Brisbane, 1995)*, volume 1029 of *Lecture Notes in Comput. Sci.*, pages 174–185. Springer, Berlin, 1996.

[23] Jovan Dj. Golić and Slobodan V. Petrović. Correlation attacks on clock-controlled shift registers in keystream generators. *IEEE Trans. Comput.*, 45(4):482–486, 1996.

[24] H. M. Gustafson, L. R. Simpson, and J. Dj. Golić. Analysis of a measure of correlation between two binary strings of different lengths. *Australas. J. Combin.*, 25:185–199, 2002.

[25] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Inform. Theory*, 26(4):401–406, 1980.

[26] Fredrik Jönsson and Thomas Johansson. A fast correlation attack on LILI-128. *Inform. Process. Lett.*, 81(3):127–132, 2002.

[27] Aviad Kipnis and Adi Shamir. Cryptanalysis of the HFE public key cryptosystem by relinearization. In *Wiener, Michael (ed.), Advances in cryptology - CRYPTO '99. 19th annual international cryptology conference Santa Barbara, CA, USA, August 15-19, 1999. Proceedings. Berlin: Springer. Lect. Notes Comput. Sci. 1666, 19-30* . 1999.

[28] Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997. With a foreword by P. M. Cohn.

[29] Alexander Maximov, Thomas Johansson, and Steve Babbage. An improved correlation attack on A5/1. In *Selected areas in cryptography*, volume 3357 of *Lecture Notes in Comput. Sci.*, pages 1–18. Springer, Berlin, 2005.

[30] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *J. Cryptology*, 1(3):159–176, 1989.

[31] C. E. Shannon. Communication theory of secrecy systems. *Bell System Tech. J.*, 28:656–715, 1949.

[32] T. Siegenthaler. Correlation-immunity of nonlinear combining functions for cryptographic applications. *IEEE Trans. Inform. Theory*, 30(5):776–780, 1984.

[33] Leonie Ruth Simpson, E. Dawson, J. Golić, and William L. Millan. LILI keystream generator. In *Selected areas in cryptography (Waterloo, ON, 2000)*, volume 2012 of *Lecture Notes in Comput. Sci.*, pages 248–261. Springer, Berlin, 2001.

[34] Douglas R. Stinson. *Cryptography.* Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, third edition, 2006. Theory and practice.